PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : G06F 9/44 — A1

(11) International Publication Number: **WO 00/23885**

(43) International Publication Date: 27 April 2000 (27.04.00)

(21) International Application Number: PCT/US99/24242

(22) International Filing Date: 15 October 1999 (15.10.99)

(30) Priority Data:
09/173,976 16 October 1998 (16.10.98) US

(71) Applicant: SOFTBOOK PRESS, INC. [US/US]; Suite 200, 1075 Curtis Street, Menlo Park, CA 94025 (US).

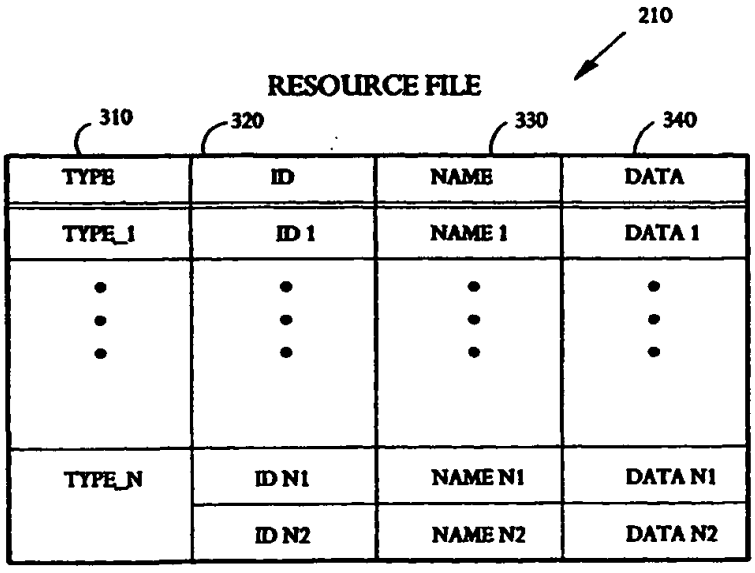
(72) Inventors: WALTER, Erik; 8288 Gilman Drive #43, La Jolla, CA 92037 (US). CONBOY, Garth; 359 Belvedere Street, La Jolla, CA 92037 (US). DUGA, Brady; 3342 Cuesta Place, Carlsbad, CA 92009 (US).

(74) Agents: BLAKELY, Roger, W. et al.; Blakely, Sokoloff, Taylor & Zafman, 7th floor, 12400 Wilshire Boulevard, Los Angeles, CA 90025–1026 (US).

(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: STORAGE OF STATIC DATA FOR EFFICIENT ACCESS AND FIELD UPGRADE



(57) Abstract

The present invention is a method and apparatus for storing static data in a memory. The static data are represented according to a resource file structure and included in a source code of an execution code. The source code is compiled to generate a machine code which includes the static data and the execution code. The machine code is transferred to the memory. The technique allows access to a data field in a data structure embedded in a program code. The data field corresponds to a structure element. A pointer to the data field which is stored in the data structure is obtained. The data field is retrieved using the pointer.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

STORAGE OF STATIC DATA FOR EFFICIENT ACCESS AND FIELD UPGRADE

BACKGROUND

1. Field of the Invention

This invention relates to data storage. In particular, the invention relates to efficient storage of static data.

2. Description of Related Art

Thanks to advances in computer and storage technologies, information retrieval programmable devices have become popular. Consumers can have portable devices to access a large amount of on-board data or information downloaded from a communication network. Examples of these information retrieval programmable devices include Personal Digital Assistant (PDA) and electronic book (EB).

There are two types of data accessible to a programmable device: static data and dynamic data. Static data are data that are not modified when the device is in use. Examples of static data include constants, tables, and fixed image structures. Dynamic data are data that are modified dynamically when the device is in use. Examples of dynamic data include graphic information and computational data structures.

Static data are usually stored on mass storage devices (e.g., disk) as a database system. When the programmable device requires the information from the database, static data are transferred to on-board random access memory (RAM) to allow the processor on the programmable device to access directly.

The use of a disk subsystem on an information retrieval programmable unit presents a number of disadvantages. First, a disk subsystem is bulky, creating inconvenience to users and difficulty in the manufacturing process. Second, a disk subsystem is slow, resulting in long start-up time when the programmable unit is powered up and initialized. Third, using RAM to store data transferred from a disk subsystem is redundant and takes up valuable RAM storage space. Fourth, it is inconvenient to upgrade the static data in the field or during repair or service. Fifth, since a disk subsystem is integrated separately from other electronic components on the programmable unit, it is uncertain if the static data stored on the disk subsystem corresponds to the executable version of the on-board program.

Therefore there is a need in the technology to provide a simple and efficient method to store static data on an information retrieval device for easy access and convenient upgrading.

SUMMARY

The present invention is a method and apparatus for storing static data in a memory. The static data are represented according to a resource file structure and included in a source code of an execution code. The source code is compiled to generate a machine code which includes the static data and the execution code. The machine code is transferred to the memory. The technique allows access to a data field in a data structure embedded in a program code. The data field corresponds to a structure element. A pointer to the data field which is stored in the data structure is

obtained. The data field is retrieved using the pointer.

BRIEF DESCRIPTION OF THE DRAWINGS

The features and advantages of the present invention will become apparent from the following detailed description of the present invention in which:

Figure 1 is a diagram illustrating a system in which one embodiment of the invention can be practiced.

Figure 2 is a diagram illustrating an environment for storing static data and field update according to one embodiment of the invention.

Figure 3 is a diagram illustrating an resource file structure of a resource file representing the static data according to one embodiment of the invention.

Figure 4 is a diagram illustrating a data structure of a resource file on a ROM/flash program code according to one embodiment of the invention.

Figure 5 is a flowchart illustrating a process of retrieving resource information according to one embodiment of the invention.

Figure 6 is an example of a resource file according to one embodiment of the invention.

Figure 7 is a machine code of the resource file shown in Figure 6 according to one embodiment of the invention.

Figure 8A is a source code to store the static data for a target processor shown in Figure 7 according to one embodiment of the invention.

Figure 8B is a source code to store the static data for a non-target processor shown in Figure 7 according to one embodiment of the invention.

DESCRIPTION

The present invention is a method and apparatus for storing static data on an information retrieval programmable device for efficient access and convenient field update. The static data are incorporated into the program code stored on a programmable or flash memory. The static data are organized as a resource file with a simple data structure. Upon initialization, the static data are accessed efficiently using this simple data structure. The static data are also conveniently updated remotely or in the field via a communication port or by a plug-in replacement of the programmable or flash memory.

In the following description, for purposes of explanation, numerous details are set forth in order to provide a thorough understanding of the present invention. However, it will be apparent to one skilled in the art that these specific details are not required in order to practice the present invention. In other instances, well-known electrical structures and circuits are shown in block diagram form in order not to obscure the present invention.

Figure 1 is a diagram illustrating a system in which one embodiment of the invention can be practiced.

Referring to Figure 1, the system 100 comprises: (a) at least one portable electronic book 10 operative to request a digital content from a catalog of distinct digital contents, to receive and display the requested

digital content in readable form; (b) an information services system 20 which includes an authentication server 32 for authenticating the identity of the requesting portable electronic book 10 and a copyright protection server 22 for rendering the requested digital content sent to the requesting portable electronic book 10 readable only by the requesting portable electronic book 10; (c) at least one primary virtual bookstore 40 in electrical communication with the information services system 20, the primary virtual bookstore being a computer-based storefront accessible by the portable electronic book and including the catalog of distinct digital contents; and (d) a repository 50, in electrical communication with the primary virtual bookstore 40, for storing the distinct digital contents listed in the catalog.

The system 100 preferably includes more than one portable electronic book 10, to be commercially viable. This is illustrated in Figure 1 by including the portable electronic books 12 and 14. The system also preferably includes more than one primary virtual bookstore 40, each serving a different set of customers, each customer owning a portable electronic book.

In one embodiment of the invention, the system 100 further comprises a secondary virtual bookstore 60 in electrical communication with the information services system 20. In this case, the information services system 20 also includes a directory of virtual bookstores 26 in order to provide the portable electronic book 10 with access to the secondary virtual bookstore 60 and its catalog of digital contents.

The information services system 20 can optionally include a notice board server 28 for sending messages

from one of the virtual bookstores, primary or secondary, to a portable electronic book in the system.

The information services system 20 also includes a registration server 24 for keeping track of the portable electronic books that are considered active accounts in the system and for ensuring that each portable electronic book is associated with a primary virtual bookstore in the system. In the case where the optional notice board server 28 is included in the information services system 20, the registration server 24 also allows each portable electronic book user to define his/her own notice board and document delivery address.

The information services system 20 preferably comprises a centralized bookshelf 30 associated with each portable electronic book 10 in the system. Each centralized bookshelf 30 contains all digital contents requested and owned by the associated portable electronic book 10. Each portable electronic book 10 user can permanently delete any of the owned digital contents from the associated centralized bookshelf 30. Since the centralized bookshelf 30 contains all the digital contents owned by the associated portable electronic book 10, these digital contents may have originated from different virtual bookstores. The centralized bookshelf 30 is a storage extension for the portable electronic book 10. Such storage extension is needed since the portable electronic book 10 has limited non-volatile memory capacity.

The user of the portable electronic book 10 can add marks, such as bookmarks, inking, highlighting and underlining, and annotations on a digital content displayed on the screen of the portable electronic book, then stores this marked digital content in the non-volatile memory of the electronic book 10. The user can

also upload this marked digital content to the information services system 20 to store it in the centralized bookshelf 30 associated with the portable electronic book 10, for later retrieval. It is noted that there is no need to upload any unmarked digital content, since it was already stored in the centralized bookshelf 30 at the time it was first requested by the portable electronic book 10.

The information services system 20 further includes an Internet Services Provider (ISP) 34 for providing Internet network access to each portable electronic book in the system.

Figure 1 further illustrates that the electronic books 10, 12, and 14 are used in the field. The electronic book 10 may be upgraded via the Internet or an upgrade ROM/ flash cartridge 270.

Figure 2 is a diagram illustrating an environment for storing static data and field update according to one embodiment of the invention. The environment 200 includes a development/manufacturing stage 205 and a field use stage 206.

The development/manufacturing stage 205 includes the development of prototypes and manufacturing of final products. During prototype development or product manufacturing, the development/manufacturing stage 205 stores static data in the information retrieval programmable device. The development/manufacturing stage 205 includes a resource file structure 210 and a read-only memory (ROM)/flash memory 220. The resource file structure 210 is a database structure that organizes the static data in a predefined format. The static data includes data whose values are not changed during program execution, or from execution to

execution. Examples of the static data include the parse tables, images, user interface layout information, string constants, etc. Although static data may be updated periodically, once they are incorporated into the information retrieval programmable device, they are not changed during the use of the device. The static data can be hard coded into a program's source code. A resource file is a database that organizes the static data in a structured manner to facilitate information access and management. The resource file structure 210 may include a number of resource files, each representing a resource type. The resource file structure 210 is incorporated into the ROM/flash memory 220.

The ROM/flash memory 220 represents a storage system in the information retrieval programmable unit. The ROM/flash memory 220 may be implemented by semiconductor memories such as programmable read-only memory (PROM) or flash memory. The ROM/flash memory 220 normally stores the program code that is executed by the processor in the information retrieval programmable unit. Storing the static data in the program code instead of a disk system provides a number of advantages. First, the ROM/flash memory 220 is more compact, light, reliable, and faster than a disk system. Static data can be retrieved quickly, reducing start-up time when the unit is initialized at power-up. Second, there is no need to use valuable random access memory (RAM) storage space to store the static data transferred from the disk system. Third, the updating of the static data can be conveniently and efficiently performed at the field merely by replacing or reprogramming the ROM/flash memory 220. Fourth, the integrity of information is maintained by storing static data together with program code in the same physical

ROM/flash memory. The synchronization of information storage is guaranteed when the ROM/flash memory 220 is programmed.

The program code in the ROM/flash 220 includes a static data structure 222 and an executing code 224. The static data structure 222 contains the static data transferred or compiled from the resource file structure 210. The static data structure 222 has a simple data structure that allows direct access by the processor. The executing code 224 is the code that is executable by the processor. The processor can access the static data structure 222 by executing the appropriate portion of the executing code 224. The executing code 224 also contains other functions, routines, subprograms, etc. that allows the processor to perform the intended tasks of information retrieval in the programmable unit.

The field use stage 206 is the stage when the unit is finalized and is used in the field, either by the user or by the field technician. The field use stage 206 includes the field unit 240. The field unit 240 represents the final product. The field unit 240 includes a programmed ROM/flash memory 250, a processor 255, and a remote port 260. The programmed ROM/flash memory 250 is the ROM/flash memory 220 that has been programmed with the static data in the development/ manufacturing stage 205. The processor 255 is any processor that can execute the code contained in the programmed ROM/flash memory 250 and access the static data structure embedded in the program code. The remote port 260 represents a communication port that allows an upgrade ROM/flash cartridge 270 to update the programmed ROM/flash memory 250. The upgrade ROM/ flash cartridge 270 contains the updated program code that replaces the program code stored in the programmed ROM/ flash memory

250. The replacement of the programmed ROM/flash memory 250 can be performed by transferring the contents of the upgrade ROM/flash cartridge 270. The transfer can be done locally or remotely via a communication network. The use of a remote upgrade provides a fast, efficient, and convenient upgrading procedure.

Figure 3 is a diagram illustrating the resource file structure 214 of a resource file representing the static data according to one embodiment of the invention.

A resource file is a hierarchical organization of resource data as static data. The static data may be organized as a multi-level data structure. In one embodiment, the resource file has two tiered data objects, or two levels. The first level is the resource type and the second level is the resource identification (ID). As illustrated in Figure 3, the resource file structure 214 has four fields: a type field 310, an ID field 320, a name field 330, and a data field 340.

The type field 310 indicates the resource type. In the example shown in Figure 3, the type field 310 has N types from TYPE_1 to TYPE_N. In one embodiment, the type field is 32-bit or 4-byte long. The ID field 320 indicates the ID code for the corresponding static data. In one embodiment, the ID field 320 is 16-bit long. A resource type may have a number of different ID's. In the example shown in Figure 3, the resource TYPE_1 has one ID, ID_1, and the resource TYPE_N has two ID's, ID_N1 and ID_N2. The name field 330 is optionally provided to give the ID a descriptive name. It is represented by a string key. In the example shown in Figure 3, the ID_1 has a NAME_1, the ID_N1 has a NAME_N1, and the ID_N2 has a NAME_N2. The data field

340 is the value of the corresponding ID. The length of the data field depends on the ID of the data. In the example shown in Figure 3, the ID_1 has DATA_1, the ID_N1 has DATA_N1, the ID_N2 has DATA_N2.

The resource file structure 210 is included as part of the source code of the program code. During development, the source code of the program code is compiled and the machine code for the program code is produced which include the corresponding static data represented as the resource file. The machine code is then programmed to the ROM/flash memory for final unit. The organization of the static data in the machine code has a simple data structure to allow fast access.

Figure 4 is a diagram illustrating a data structure 400 of a resource file on a ROM/flash program code according to one embodiment of the invention.

The data structure 400 of the embedded static data as compiled from the source code follows the resource file structure 210. However, to facilitate the access and management, additional fields are provided. In particular, two additional fields are defined: the name length field and the data length field. The name length field specifies the length of the name. The data length field specifies the length of the data field. Using the lengths of the name and data fields, it is possible to determine the address (or pointer) of any item in the resource file.

The data structure 400 includes a version stamp 410, type blocks 4201 through 420N, and an end of resource file indicator 430.

The version stamp 410 is the version of the resource file to provide a track history for the

resource file. The version stamp 410 provides a means for record keeping and for field upgrade.

The type blocks 4201 to 420N are the static data organized according to TYPE. Each type block has three major parts: a TYPE part, a DESCRIPTOR part, and an END_OF_TYPE part. The DESCRIPTOR part contains the fields as described in Figure 3 with the addition of two fields: the NAME_LENGTH and DATA_LENGTH fields which show the length of the name and the data, respectively. In the example shown in Figure 4, the type block 4201 has a part TYPE_1 4301, the DESCRIPTOR_1 part 4401, and an END_OF_TYPE_1 part 4601. The DESCRIPTOR_1 part 4401 includes the following fields: ID_1 4421, name length_1 4441, NAME_1 446, DATA LENGTH_1 4481, DATA_1 4501, and END_OF_TYPE_1 4601. The type block 420N has a TYPE_N part 430N, a DESCRIPTOR_N1 part 440N1, a DESCRIPTOR_N2 part 440N2, and an END_OF_TYPE_N part 460N. The DESCRIPTOR_N1 part 440N1 has the following fields: ID_N1 442N1, NAME_LENGTH_N1 444N1, NAME_N1 446N1, DATA_LENGTH_N1 448N1, and DATA_N1 450N1. The DESCRIPTOR_N2 part 440N2 has the following fields: ID_N2 442N2, NAME_LENGTH_N2 444N2, NAME_N2 446N2, DATA_LENGTH_N2 448N2, and DATA_N2 450N2.

The end of resource file indicator 430 is a delimiter code to indicate the end of the resource file.

Figure 5 is a flowchart illustrating a process 500 of retrieving resource information according to one embodiment of the invention.

Upon START, the process 500 obtains the pointer for the resource file (Block 510). The pointer is the address offset stored in some fixed location or part of the execution code to allow the processor to point to the beginning of the data structure. Then the process

500 obtains the number of types in the resource file (Block 520), and the number of ID's in each type (Block 530). Then the process 500 obtains the data lengths of the ID's in each type (Block 540).

Using the number of types and the number of ID's in each type, the process 500 determines the pointer for the ID k in TYPE j (Block 550). Then the process 500 accesses the resource file for the specified type and ID using the pointer computed in block 550 (Block 560). Then the process 500 is terminated.

Figure 6 is an example of a resource file according to one embodiment of the invention.

The resource file 600 has a name foo.RES. There are two types in the "foo.RES" resource file 600: TYPE and TTOO. The TYPE type has two resource ID's: 1001 and 1002. The resource ID 1001 has a name "Name1" and a data "Data1". The resource ID 1002 has a name "Name2" and a data "Data2". The TTOO type has one resource ID: -31782. The resource ID -31782 has an empty name "" and a data "Data3".

Figure 7 is a binary data 700 of the resource file shown in Figure 6 according to one embodiment of the invention.

The binary data 700 represents the static data in hexadecimal of the resource file shown in Figure 6. The binary data 700 includes an address field 710 and a content field 720. The address and content fields are in hexadecimal.

The address field 710 shows the addresses of the corresponding static data items. Address location 00 contains the version number. In this example, the version number is 0001. Address location 02 contains

the resource type TYPE encoded as "54 59 50 45". Address location 06 contains dummy information reserved for future use. Address location 0A contains the offset for the resource file. This offset is used as a pointer to point to the logical beginning of the static data structure. The logical beginning of the static data structure is at address location 2A. Address location 0E contains dummy data, reserved for other uses.

Address location 20 contains the value of "Data2" having 5 bytes "44 61 74 61 32". Address location 25 contains the value of "Data1" having 5 bytes "44 61 74 61 31".

Address location 2A contains the ID number 1002 or "03 EA" in hexadecimal. Address location 2C contains the size of the data of ID number 1002 which is "Data2". The size of "Data2" is 5, so address location 2C contains "00 00 00 05". Address location 30 contains the offset or pointer to point to "Data2". Since "Data2" is stored at address location 20, address location 30 contains "00 00 00 20". Address location 34 contains the size of the resource name "Name2". The resource name "Name2" has 5 bytes, so address location 34 contains "00 05". Address location 36 contains the code for "Name2" which is 5-byte long, "4E 51 6D 65 32".

Address location 3B contains the ID number 1001 or "03 E9" in hexadecimal. Address location 3D contains the size of the data of ID number 1001 which is "Data1". The size of "Data1" is 5, so address location 3D contains "00 00 00 05". Address location 41 contains the offset or pointer to point to "Data1". Since "Data1" is stored at address location 25, address location 41 contains "00 00 00 25". Address location 45 contains the size of the resource name "Name1". The resource name "Name1" has 5 bytes, so address location

45 contains "00 05". Address location 47 contains the code for "Name1" which is 5-byte long, "4E 51 6D 65 31".

Figure 8A is a source code to store the static data for a target processor shown in Figure 7 according to one embodiment of the invention.

The first version, compiled under the compiler directive #if defined(_MC68K_) is for the target programmable device platform. In this example, it is a Motorola 68000 family processor. This version is coded as a "procedure" having a collection of in-line functions. The code is compiled and linked directly into a ROM/flash storable and executable module.

Figure 8B is a source code to store the static data for a non-target processor shown in Figure 7 according to one embodiment of the invention.

The second version, indicated by the compiler directive #if not defined(_MC68K_), is built as initialized data for use on non-target platforms.

The present invention provides a simple and efficient technique to store static data on an information retrieval programmable device. The static data are organized as a resource file which is compiled together with the program code and stored on a ROM/ flash memory. The technique allows fast access and convenient field update.

While this invention has been described with reference to illustrative embodiments, this description is not intended to be construed in a limiting sense. Various modifications of the illustrative embodiments, as well as other embodiments of the invention, which are apparent to persons skilled in the art to which the

invention pertains are deemed to lie within the spirit and scope of the invention.

CLAIMS

What is claimed is:

1. A method for storing static data in a memory, the method comprising:

(a) representing the static data according to a resource file structure;

(b) including the represented static data in a source code of an execution code;

(c) compiling the source code to generate a machine code, the machine code including the static data and the execution code; and

(d) transferring the machine code to the memory.

2. The method of claim 1 wherein the resource file structure has a hierarchical organization.

3. The method of claim 2 wherein the resource file structure includes a type field, an ID field, a name field, and a data field.

4. The method of claim 3 wherein resource file structure further includes a name length field and a data length field.

5. The method of claim 1 herein compiling includes compiling a target version and a non-target version.

6. The method of claim 5 wherein the target version corresponds to a target processor platform.

7. The method of claim 6 wherein the non-target version corresponds to a non-target platform.

8. A method to access a data field in a data structure embedded in a program code, the data field corresponding to a structure element, the method comprising:

(a) obtaining a pointer to the data field corresponding to the structure element, the pointer being stored in the data structure; and

(b) retrieving the data field using the pointer.

9. The method of claim 8 wherein the structure element has a hierarchical organization.

10. The method of claim 9 wherein the structure element includes a type field and an identification field.

11. The method of claim 10 wherein the structure element further includes a name length field and a data length field.

12. The method of claim 9 wherein the structure element includes a type field and an identification field.

13. An apparatus comprising:

a programmed memory that stores a static data structure and an execution code, the static data structure including a data field, the execution code referencing the static data structure; and

a processor coupled to the memory for executing the execution code to obtain a value of the data field.

14. The apparatus of claim 13 further comprising:

an interface port coupled to the processor and the memory for interfacing to an upgrade cartridge, the upgrade cartridge containing an upgrade memory.

15. The apparatus of claim 14 wherein the processor transfers content of the upgrade memory to the programmed memory when the programmed memory needs updating.

16. The apparatus of claim 13 wherein the static data structure includes a type field, an ID field, a name field, and a data field.

17. The apparatus of claim 16 wherein the static data structure further includes a name length field and a data length field.

18. The apparatus of claim 17 wherein the static data structure further includes a data pointer pointing to the data field.

19. The apparatus of claim 18 wherein the processor obtains the value of the data field using the data pointer.

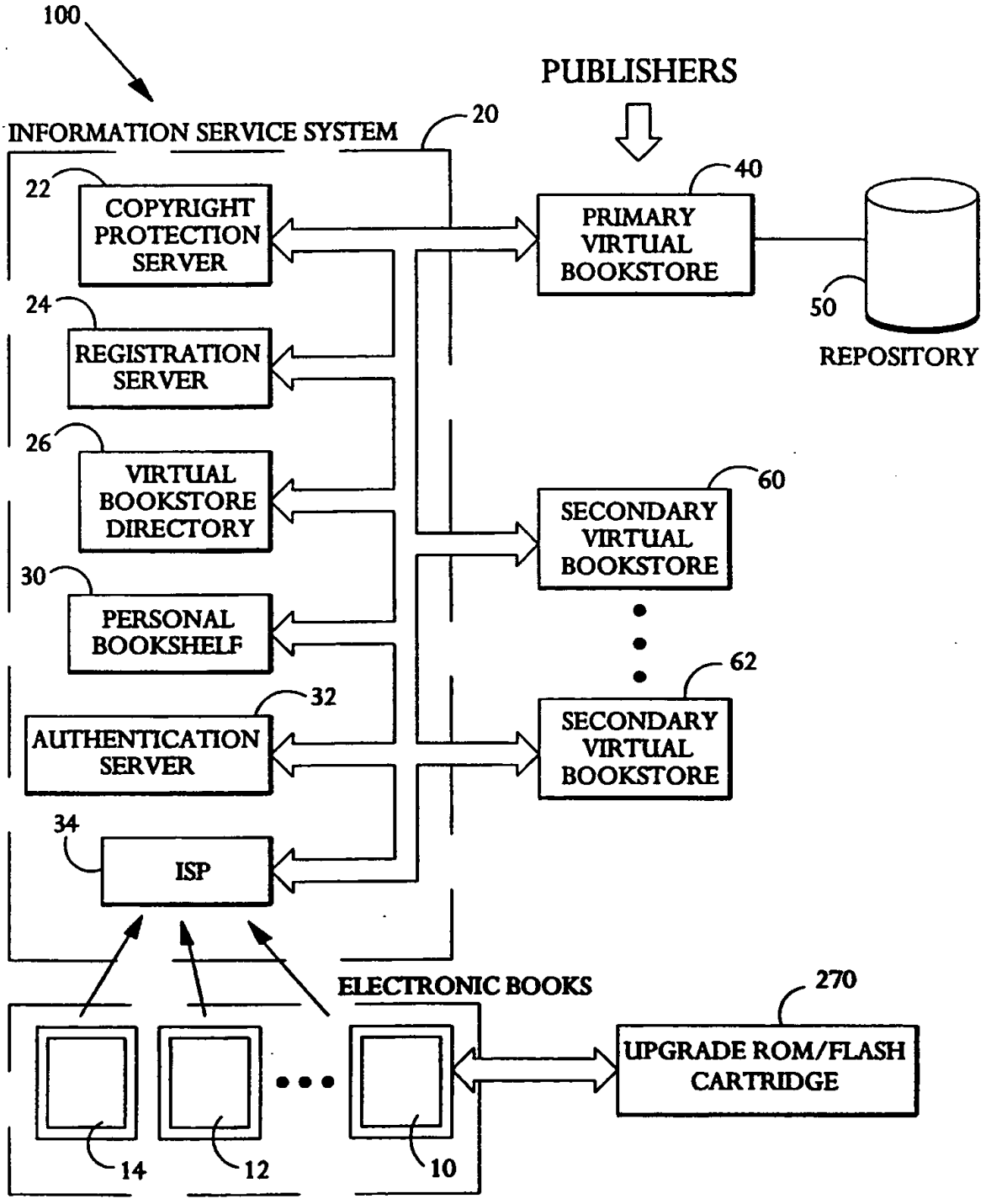

100
PUBLISHERS
INFORMATION SERVICE SYSTEM
20
22
COPYRIGHT PROTECTION SERVER
24
REGISTRATION SERVER
26
VIRTUAL BOOKSTORE DIRECTORY
30
PERSONAL BOOKSHELF
32
AUTHENTICATION SERVER
34
ISP
40
PRIMARY VIRTUAL BOOKSTORE
50
REPOSITORY
60
SECONDARY VIRTUAL BOOKSTORE
62
SECONDARY VIRTUAL BOOKSTORE
ELECTRONIC BOOKS
270
UPGRADE ROM/FLASH CARTRIDGE
14
12
10

Fig. 1

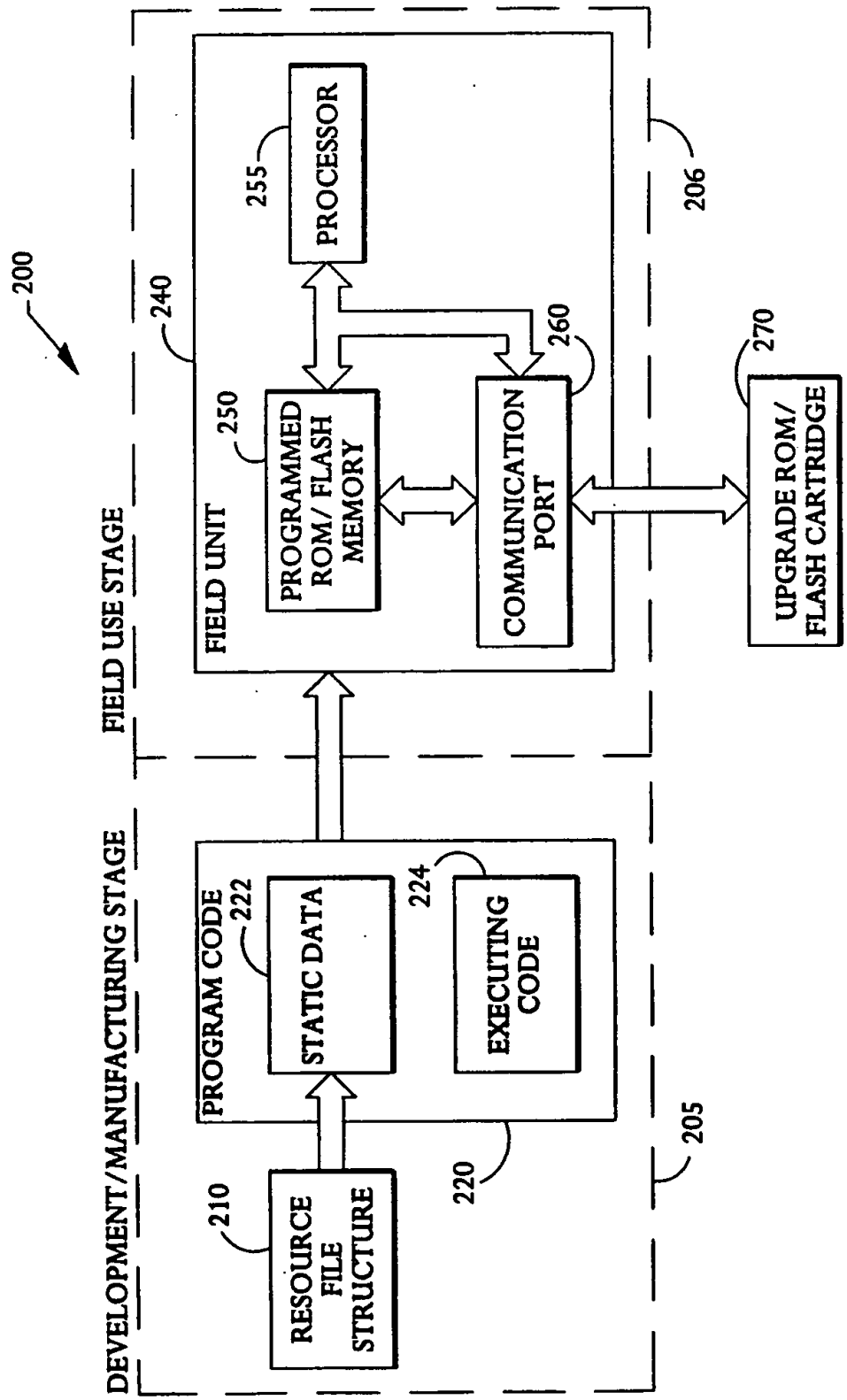

200
DEVELOPMENT/MANUFACTURING STAGE
FIELD USE STAGE
205
206
210
RESOURCE FILE STRUCTURE
220
PROGRAM CODE
222
STATIC DATA
224
EXECUTING CODE
240
FIELD UNIT
250
PROGRAMMED ROM/ FLASH MEMORY
255
PROCESSOR
260
COMMUNICATION PORT
270
UPGRADE ROM/ FLASH CARTRIDGE

Fig. 2

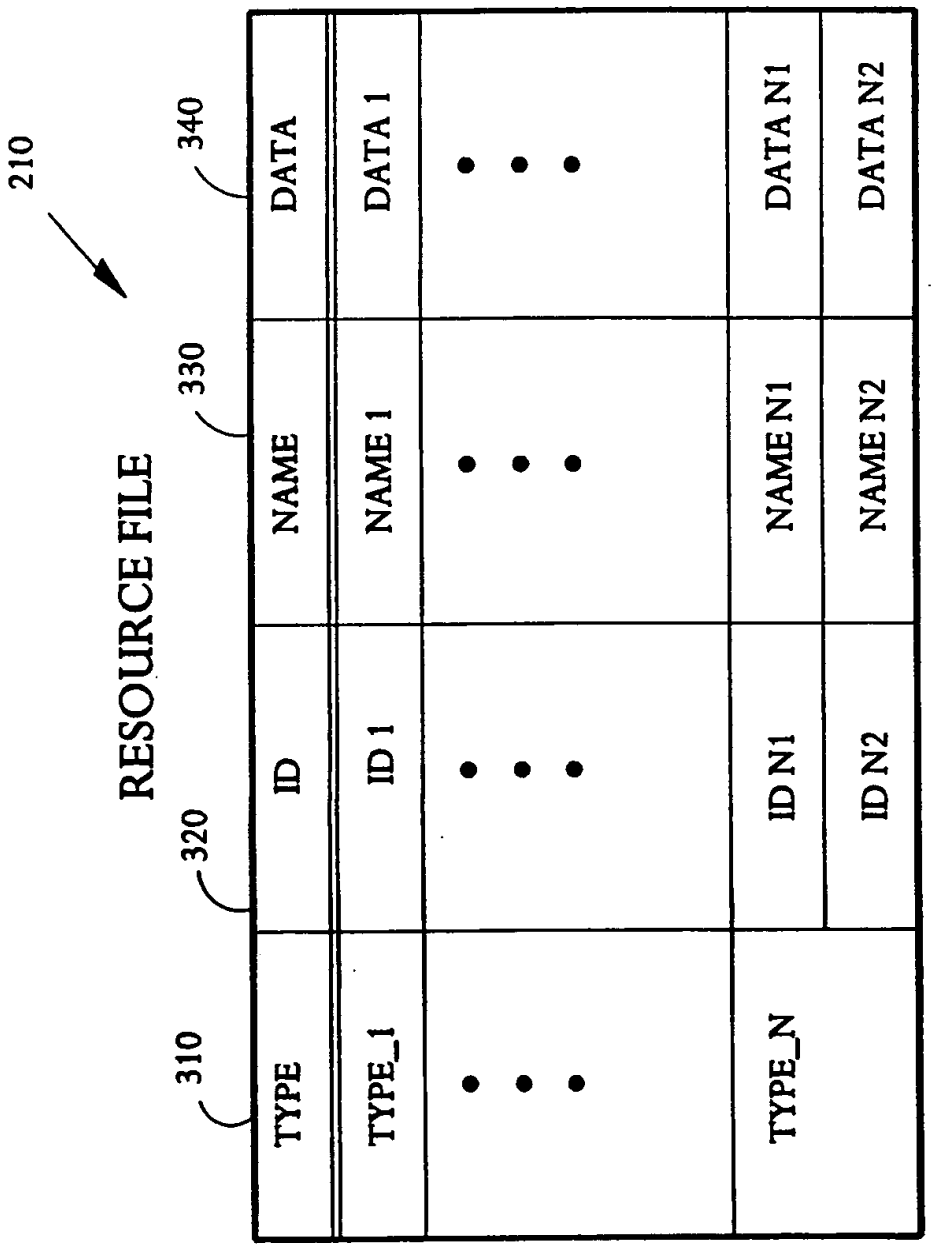

210
RESOURCE FILE
310
320
330
340
TYPE
ID
NAME
DATA
TYPE_1
ID 1
NAME 1
DATA 1
TYPE_N
ID N1
NAME N1
DATA N1
ID N2
NAME N2
DATA N2

Fig. 3

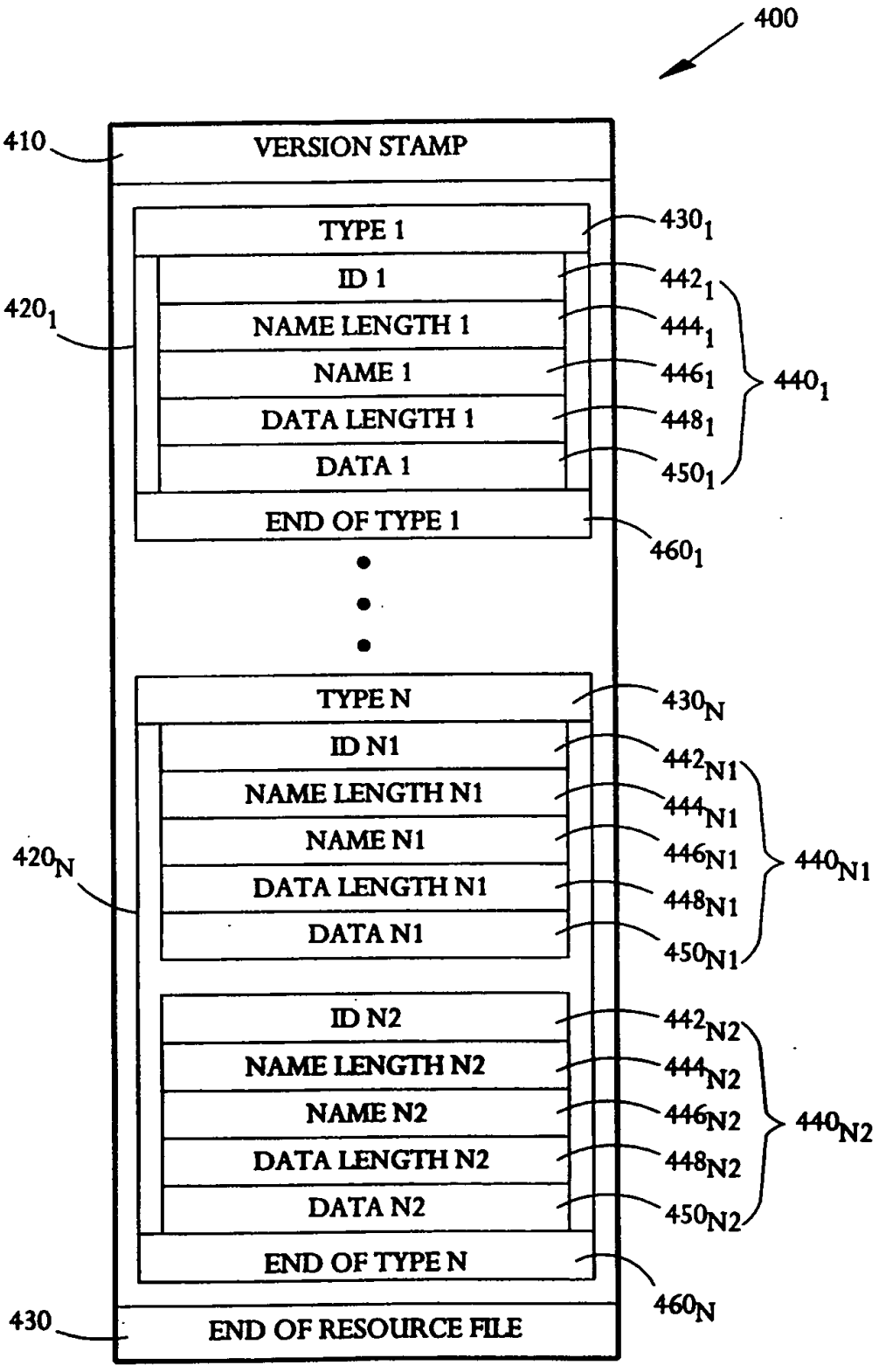

400
410
VERSION STAMP
TYPE 1
ID 1
NAME LENGTH 1
NAME 1
DATA LENGTH 1
DATA 1
END OF TYPE 1
TYPE N
ID N1
NAME LENGTH N1
NAME N1
DATA LENGTH N1
DATA N1
ID N2
NAME LENGTH N2
NAME N2
DATA LENGTH N2
DATA N2
END OF TYPE N
END OF RESOURCE FILE
430
420_1
420_N
430_1
442_1
444_1
446_1
448_1
450_1
440_1
460_1
430_N
442_{N1}
444_{N1}
446_{N1}
448_{N1}
450_{N1}
440_{N1}
442_{N2}
444_{N2}
446_{N2}
448_{N2}
450_{N2}
440_{N2}
460_N

Fig. 4

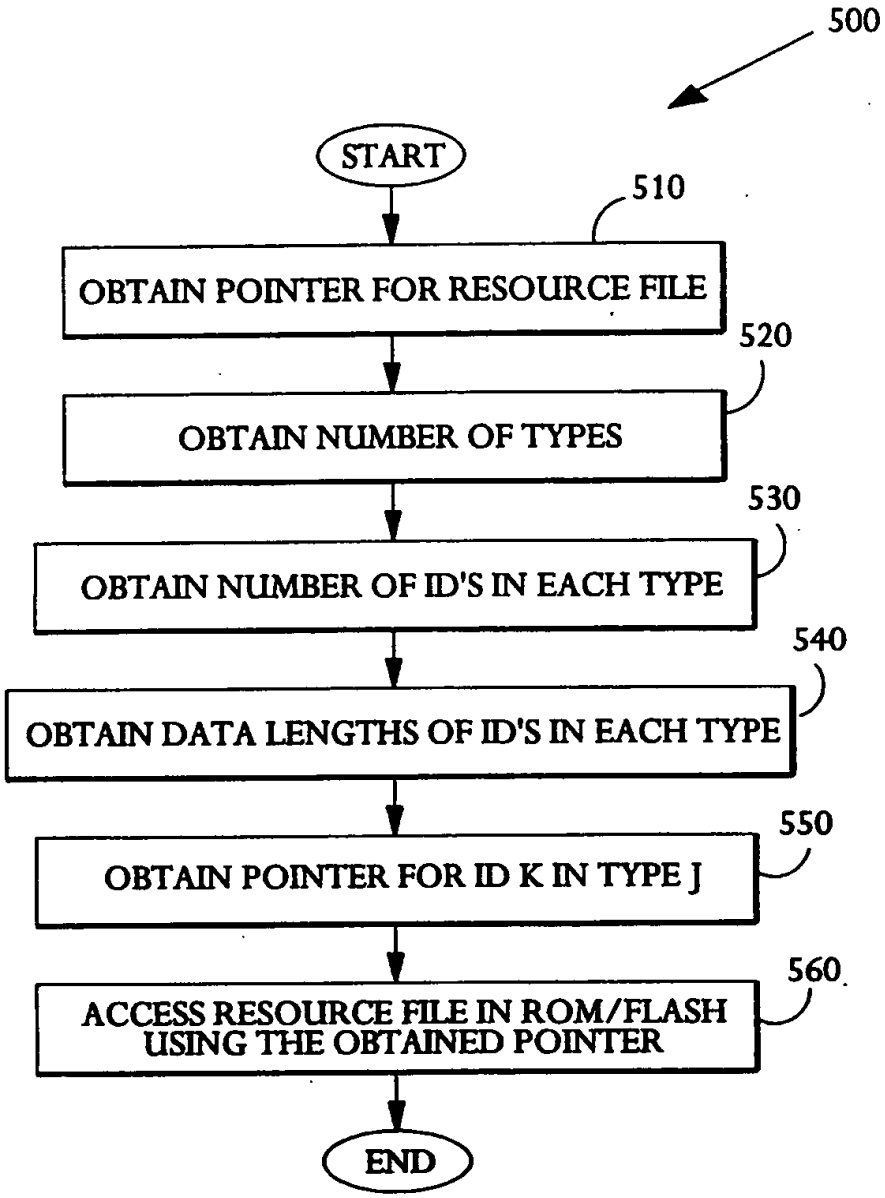

500
START
510
OBTAIN POINTER FOR RESOURCE FILE
520
OBTAIN NUMBER OF TYPES
530
OBTAIN NUMBER OF ID'S IN EACH TYPE
540
OBTAIN DATA LENGTHS OF ID'S IN EACH TYPE
550
OBTAIN POINTER FOR ID K IN TYPE J
560
ACCESS RESOURCE FILE IN ROM/FLASH USING THE OBTAINED POINTER
END

Fig. 5

Resource File	Resource Type	Resource ID	Resource Name	Resource Data
foo.RES				
	TYPE			
		1001	"Name1"	"Data1"
		1002	"Name2"	"Data2"
	TTOO			
		-31782	""	"Data3"

Fig. 6

```
00:     00 01                   - Version stamp
02:     54 59 50 45             - Resource type: TYPE
06:     00 00 00 00             - Storage bytes "wasted"
0A:     00 00 00 2A             - Resource map offset
0E:     <18 padding bytes>      - Padding
20:     44 61 74 61 32          - Resource data: "Data2"
25:     44 61 74 61 31          - Resource data: "Data1"
2A:     03 EA                   - Resource ID: 1002
2C:     00 00 00 05             - Data size: 5
30:     00 00 00 20             - Data offset
34:     00 05                   - Resource name size: 5
36:     4E 51 6D 65 32          - Resource name: "Name2"
3B:     03 E9                   - Resource ID: 1001
3D:     00 00 00 05             - Data size: 5
41:     00 00 00 25             - Data offset
45:     00 05                   - Resource name size: 5
47:     4E 61 6D 65 31          - Resource name: "Name1"
```

Fig. 7

```
#if defined(__MC68K__)

extern "C"
{
    static void Routine1(void) =
    {
        0x0000, 0x0001,                     // Version stamp.
        //  'T',  'T',  'O',  'O'          // Resource type.
        0x5454, 0x4F4F,
        0x83DA,                             // Resource ID.
        0x0000,                             // Resource name length.
        //  0x00, 0x00
        0x0000,
        //  0x00, 0x00, 0x00, 0x05          // Data length.
        0x0000, 0x0005,
        //  'D',  'a',  't',  'a',  '3',  0x00
        0x4461, 0x7461, 0x3300,
        //  0x80, 0x00                      // End of resource type.
        0x8000,
        //  'T',  'Y',  'P',  'E'           // Resource type.
        0x5459, 0x5045,
        0x03E9,                             // Resource ID.
        //  0x00, 0x05                      // Resource name length.
        0x0005,
        //  'N',  'a',  'm',  'e',  '1',  0x00
        0x4E61, 0x6D65, 0x3100,
        //  0x00, 0x00, 0x00, 0x05          // Data length.
        0x0000, 0x0005,
        //  'D',  'a',  't',  'a',  '1',  0x00
        0x4461, 0x7461, 0x3100,
        //  0x03, 0xEA                      // Resource ID.
        0x03EA,
        //  0x00, 0x05                      // Resource name length.
        0x0005,
        //  'N',  'a',  'm',  e',  '2',  0x00
        0x4E61, 0x6D65, 0x3200,
        //  0x00, 0x00, 0x00, 0x05          // Data length.
        0x0000, 0x0005,
        //  'D',  'a',  't',  'a',  '2',  0x00
        0x4461, 0x7461, 0x3200,
        //  0x80, 0x00                      // End of resource type.
        0x8000,
        //  0xFF, 0xFF, 0xFF, 0xFF          // End of resource file data.
        0xFFFF, 0xFFFF,
        0x0000
    };
    static void Dummy(void)
    {
        Routine1();
    }
    Byte *FooData = (Byte *)Dummy + 4;
}

#endif
```

Fig. 8a

```
#if not defined(__MC68K__)

extern "C"
{
     static Byte Dummy[] =
     {
          0x00, 0x00, 0x00, 0x01,          // Version stamp.
          'T',  'T',  'O',  'O',           // Resource type.
          0x83, 0xDA,                      // Resource ID.
          0x00, 0x00,                      // Resource name length.
          0x00, 0x00,
          0x00, 0x00, 0x00, 0x05,          // Data length.
          'D',  'a',  't',  'a',  '3',  0x00,
          0x80, 0x00,                      // End of resource type.
          'T',  'Y',  'P',  'E'            // Resource type.
          0x03, 0xE9,                      // Resource ID.
          0x00, 0x05,                      // Resource name length.
          'N',  'a',  'm',  'e',  '1',  0x00,
          0x00, 0x00, 0x00, 0x05,          // Data length.
          'D',  'a',  't',  'a',  0x00,
          0x03, 0xEA,                      // Resource ID.
          0x00, 0x05,                      // Resource name length.
          'N',  'a',  'm',  'e',  '2',  0xD0,
          0x00, 0x00, 0x00, 0x05,          // Data length.
          'D',  'a',  't',  'a',  '2',  0x00,
          0x80, 0x00,                      // End of resource type.
          0xFF, 0xFF, 0xFF, 0xFF,          // End of resource file data.
          0x00
     };
     Byte *FooData = Dummy;
}

#endif
```

Fig. 8b

INTERNATIONAL SEARCH REPORT

Inte .onal Application No
PCT/US 99/24242

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F9/44

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	WO 98 06033 A (AGRANAT SYSTEMS INC) 12 February 1998 (1998-02-12) page 2, line 1 -page 3, line 2 page 3, line 30 -page 4, line 7 page 14, line 1 -page 15, line 30; figure 10	1-13, 16-19
X	CAROLINE ROSE ET AL: "Inside MacIntosh; Volumes I, II and III" October 1991 (1991-10) , ADDISON-WESLEY PUBLISHING CO. XP002131215	8-13, 16-19
A	page I-103, line 1 -page I-107, line 6 page I-131; figure 10	1-7

-/--

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
22 February 2000	09/03/2000

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Fournier, C

1

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Inta ional Application No
PCT/US 99/24242

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
Y	"OS/2 Version 2.0 ; Volume 4: Application Development" April 1992 (1992-04) , IBM INTERNATIONAL TECHNICAL SUPPORT CENTERS , USA XP002131216 page 52, paragraph 4.4 -page 53, paragraph 4.5	1-13, 16-19
Y	"OS/2 2.0 Technical Library: Presentation Manager Programming Reference, Vol. III" 1992 , IBM CORP. , USA XP002131217 page 32-1, line 1 -page 32.5, line 1	1-13, 16-19

1

INTERNATIONAL SEARCH REPORT

information on patent family members

Inte. onal Application No
PCT/US 99/24242

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9806033 A	12-02-1998	EP 0853788 A	22-07-1998
		JP 11514769 T	14-12-1999
		US 5973696 A	26-10-1999